**Mini-Project (2)**

**Step 1: Design & Implement our protocol**

**Introduction**:

Our group built the protocol on the top of UDP socket connection. We implemented connection-oriented, reliable, pipelined protocol which includes flow control and congestion control mechanisms as well. Since UDP does not inherently provide these features, we added mechanisms to ensure reliable communication, similar to TCP.

**Connection-oriented and reliable communication**: We simulate a **3-way TCP handshake** on top of the UDP protocol to make our protocol connection-oriented. We have a packet class which simlates the TCP packet structure.

Firstly, we create a UDP socket between the client and the server. Then, the client will make a packet with initial sequence number x, set the TCP SYN bit to true, and send the packet to the server. Once the server receives this packet, the server will make a packet with setting TCP SYN bit and ACK bit to true, indicating it is SYNACK, and an initial sequence number decided by the server, an ack number which is x+1.

Next, the server will send the packet it made to the client. When the client receives this packet indicating SYNACK, it shows the server is alive, the connection on the client side is established. The client then will make a packet with ack number to ack the SYNACK, send this packet to the server. If the server receives the correct ACK number, it indicates the client is alive and the connection is established.

Additionally, a timer is set for the SYN packet sent by the client and the SYNACK packet sent by the server. If either of these packets is not acknowledged within the timeout period, it will be resent for maximum 3 times, ensuring reliable communication during the handshake.

**Pipelined protocol:**

We chose to implement **Go-back-N** to ensure data reliability.

**Sender**:

1. Sender can have up to N (window size) un-acked packets in pipeline, representing the number of unacked packets in the pipeline.

2. Sender sets timer for oldest un-acket packet. When timer expires, sender will retransmit all un-acked packets.

3. Sends packets sequentially, keep track of the sequence number. The next sequence number is the current sequence number plus the length of its payload.

**Receiver**:

1. Only sends cumulative ack for the last packet received in order.

2. Does not ack packet if there’s a gap, discard out of order packets.

2. Will resend the last ack number to sender if there is a gap.

**Flow control:**

Our protocol implements flow control using receiver window size (rwnd). The receiver controls sender, and sender will not overflow. The receiver advertises its available buffer size (rwnd) in every acknowledgment packet header.The sender adjusts its transmission rate to make sure the number of outstanding packets does not exceed the receiver’s rwnd, so that we ensure the sender does not overwhelm the receiver.

**Congestion control:**

We simulate the **TCP Tahoe** congestion control.

**Slow start:** The congestion window (cwnd) starts at 1 packet and doubles the size with every successful acknowledgment until it reaches the threshold (ssthresh) or a packet loss is detected.

**Congestion avoidance**: When cwnd exceeds ssthresh, the window size grows linearly instead of exponentially.

**Loss indicated by timeout or 3 duplicate acks:** When detect packet loss, the cwnd is reset to 1 and ssthresh is set to half of the current cwnd.

**Socket type:** UDP socket, based on UDP, build TCP like protocol.

**Packet structure:** A Packet class was implemented to simulate a TCP-like packet.

Each packet contains:

* Sequence number: For tracking order.
* Acknowledgment number: For reliable communication.
* Flags: SYN, ACK, FIN flags for connection control.
* Payload: the data being transmitted. We will get the payload data from a text file.
* Receiver window size (rwnd): Advertised by the receiver for flow control.

**Step 2: Analyze Traffic**

**Simulate packet loss:**

Where to do it: in the function that sends packets, before sending packets to the receiver.

How to do it: we used a random function to simulate packet loss by randomly decide whether or not to drop a packet. We set the loss rate is 25%, if generating a random number that is less than 0.25, we will drop this packet. If the generated number is greater than 0.25, than we will send this packet over network.

A screen shot of a computer program

Description automatically generated

Sample output of simulating packet loss:

A screenshot of a computer

Description automatically generated

**Proof of connection-oriented:**

In Wireshark:

The first 3 packets are the packets being sent in the handshake part, these packets only have 17 bytes (header) and no payload. The first packet is from sender and includes TCP SYN message, the second packet is from receiver and includes SYNACK, and the third packet is from sender and includes ACK number.

A screenshot of a computer

Description automatically generated

The screenshot shows a connection-oriented protocol through the exchange of ACK numbers during the handshake process. The sender sends a SYN packet with a sequence number, and the receiver replies with an ACK (SYNACK), acknowledging the client’s sequence number and sending its own. Then, sender sends an ACK to acknowledge SYNACK. This back-and-forth of sequence and acknowledgment numbers confirms reliable communication, typical of connection-oriented protocols like TCP.

A computer screen shot of a program

Description automatically generated

**Proof of flow control:**

In receiver: set buffer size and used buffer size, put rwnd in the packet header and send it to sender:

A screenshot of a computer program

Description automatically generated

In sender: get the rwnd from the receiver, set the window size to rwnd or cwnd (smaller one), so that the sender won’t overwhelm receiver:

A screenshot of a computer program

Description automatically generated

Sample output: the sender sets the window size to receiver window size

A screenshot of a computer

Description automatically generated

**Proof of congestion control:**

At first, sender adjust the window size by slow start

A screenshot of a computer program

Description automatically generated

When the window size is greater or equal to ssthresh, the congestion avoidance happens, window size changes to increase linearly

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

When timeout or sender receive duplicate acks for 3 times, the ssthresh will be halved down, and the window size will be reset to 1, and do slow start again:

A screenshot of a computer

Description automatically generated

In Wireshark, it shows the timeout happens, it did congestion control:

A screenshot of a computer

Description automatically generated